Les coprocesseurs Xeon Phi
Architecture et modèles de programmation

kamel.mazouzi@univ-fcomte.fr

http://meso.univ-fcomte.fr

Mésocentre de calcul Franche-Comté
Université de Franche-Comté

4 décembre 2015



mésocentre de calcul de franche-comté

Plan

1. Introduction
2. Architecture
3. Modèles de programmation
4. Performances

1 Introduction

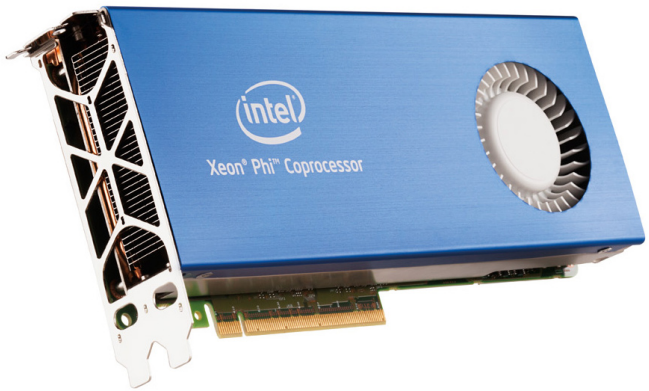
2 Architecture

3 Modèles de programmation

4 Performances

Xeon Phi - MIC - KNC

- Xeon Phi premier produit de Intel pour les architectures many-cores
- Coprocesseur
  - carte PCI express
  - doté d'un système Linux
- Dense, processeur simplifié
  - Unité vectorielle plus large
  - Plus de Threads matériels
- Plusieurs noms
  - *Many Integrated Core architecture, aka MIC*
  - *Knights Corner, aka KNC (code name)*



1 Introduction

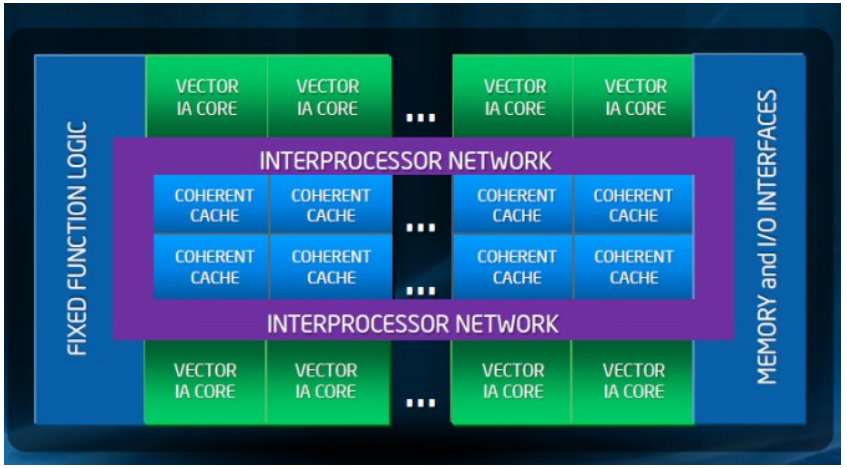
2 Architecture

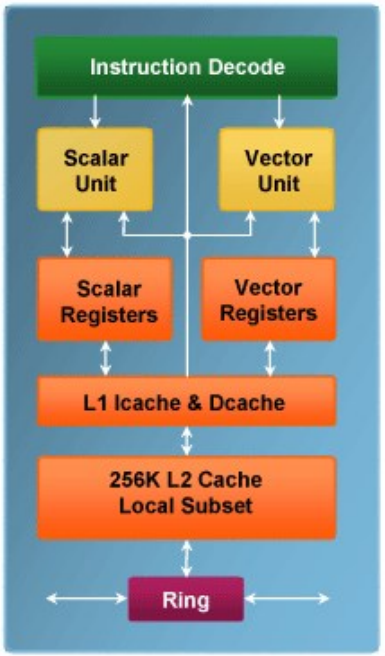
3 Modèles de programmation

4 Performances

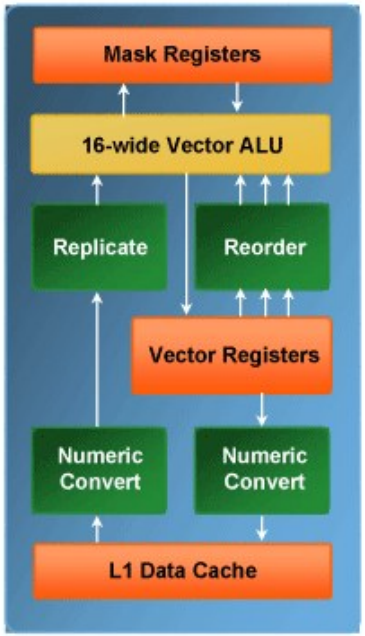
Xeon Phi - MIC

- Plusieurs cœurs, x86, spécialisés ≈ 60
- Cache L1, L2
- Anneau bidirectionnel pour le cache L2
- Accès mémoire (de la carte) très rapide
- Registres SIMD plus larges
- 4 hardware threads/coeur





Processor Core



Vector Unit

Xeon Phi vs Sandy Bridge

Xeon Phi 7120P

- Processeur
  - **61 cœurs**
  - 1.28 GHz
  - 512-bit SIMD
  - **1.2 TFlop/s** (DP)
- Cache
  - L1 32KB/cœur
  - L2 512KB/cœur, 30.5 MB/chip
- Mémoire
  - **16 GB GDDR5 DRAM**
  - **352 GB/s Peak**
- PCIe
  - 5 GB/s
  - 16 bit

Sandy Bridge E5-2680

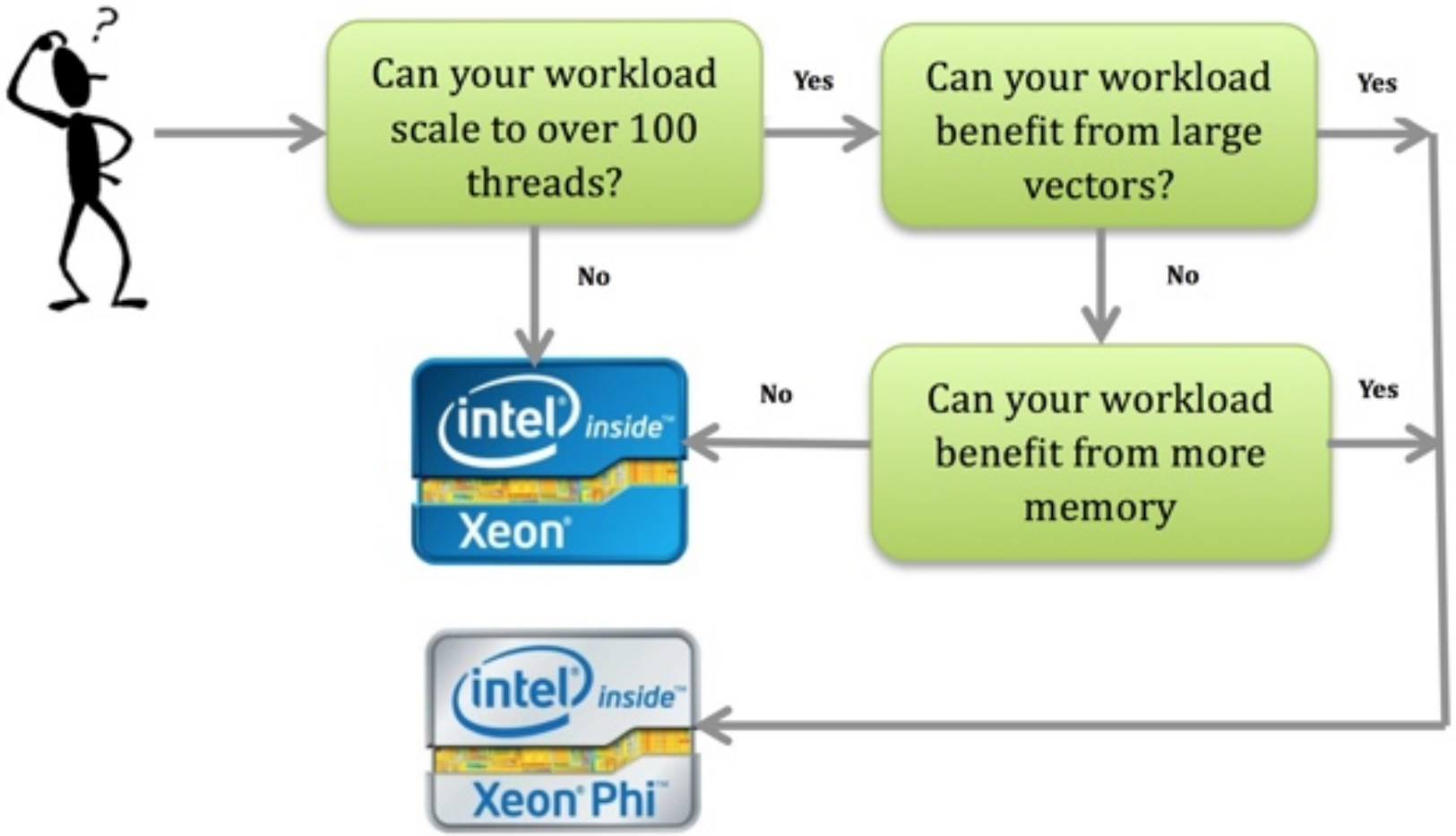
- Processeur
  - **16 cœurs**
  - 2.2 GHz
  - 256-bit SIMD
  - **250 GFlop/s** (DP)
- Cache
  - L1 32KB/cœur
  - L2 256KB/cœur
  - L3 30MB
- Mémoire
  - **64 GB DDR3**
  - **51.2 GB/s** par socket

Spécificités

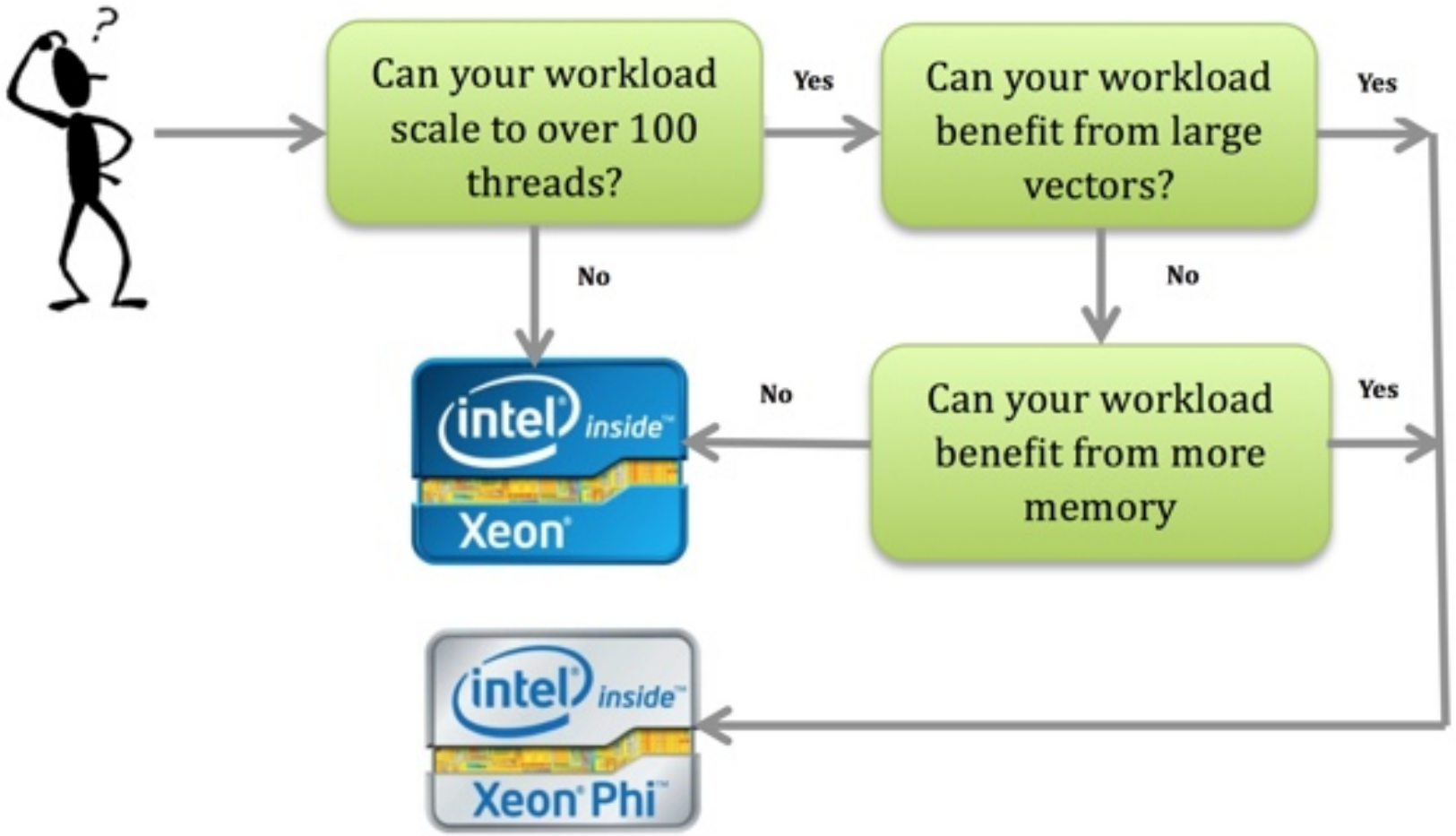
- Les Xeon Phi sont basés sur la technologie **x86**
  - plus de cœurs
  - plus de SIMD
- Même modèle de programmation que les CPUs
  - Langages : C/C++, Fortran
  - Parallélisme : OpenMP, MPI, OpenCL
  - MPI sur le nœud et sur les cartes
  - Tout code peut être exécuté sur les cartes (pas seulement les *kernels*)
- Même options d'optimisation que sur les CPUs
  - **Optimize once, run anywhere**
- Compilateurs Intel 13+ et GCC 5.1 (expérimental)
- Bibliothèques scientifiques : MKL, MAGMA MIC



Utilisation



Utilisation



Utilisation intensive des Threads et de la Vectorisation

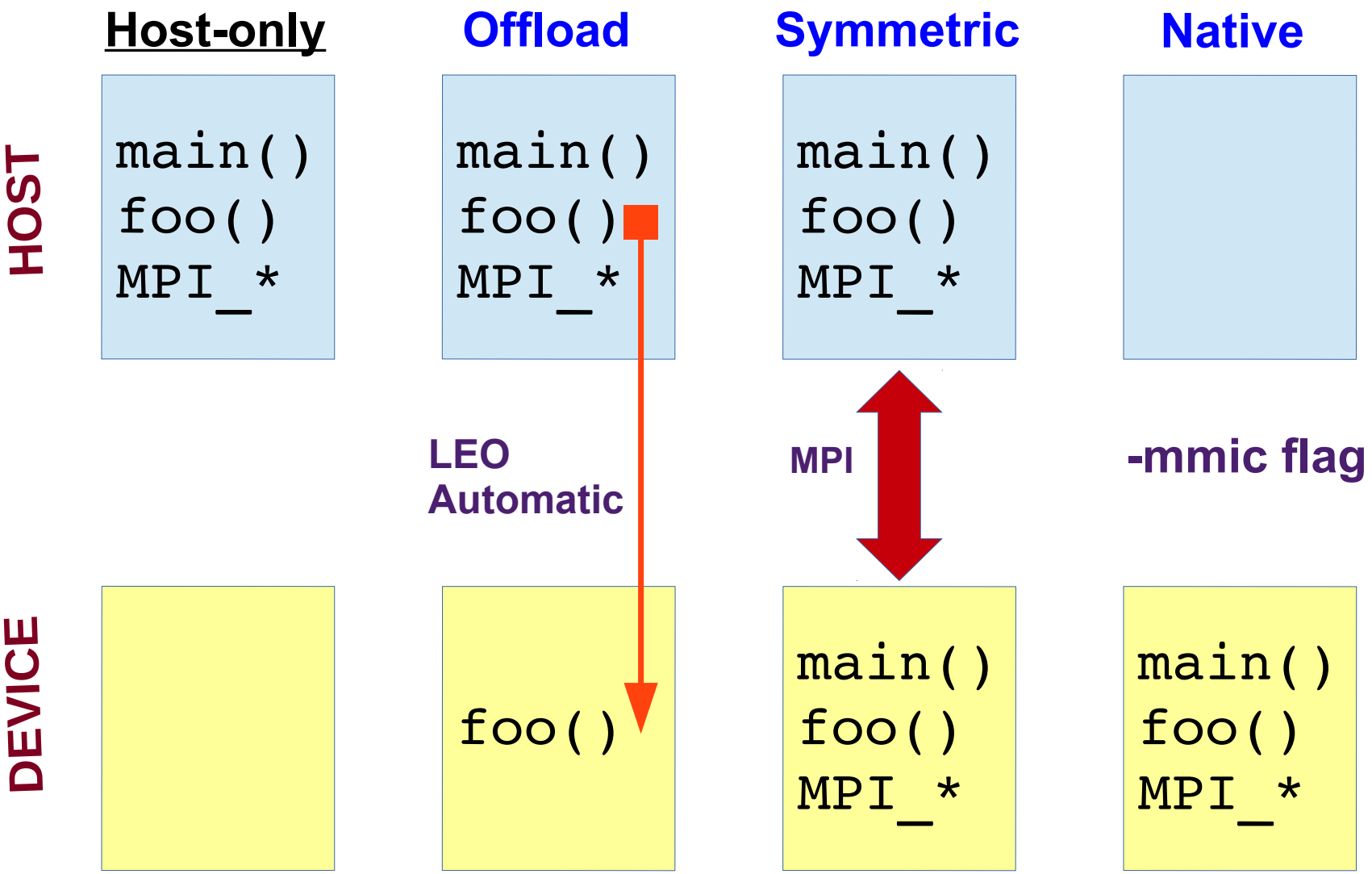
1 Introduction

2 Architecture

3 Modèles de programmation

4 Performances

Les modèles de programmation



LEO : Language Extentions for Offload

Exemple : Hello OpenMP

```
1
2  #include <stdio.h>
3  #include <omp.h>
4
5  main ()  {
6
7  int tid;
8
9  #pragma omp parallel private(tid)
10  {
11
12     tid = omp_get_thread_num();
13     printf("Hello World from thread = %d\n", tid);
14
15 }
```

Hello.c

Le mode : *native*

Depuis le noeud

- Compiler avec l'option -mmic
- Utiliser l'outil micnativeloadex pour lancer le programme
- ...ou se connecter en ssh sur la carte

```
$ icc -openmp -mmic hello.c -o hello.mic
$ micnativeloadex ./hello.mic
```

Le mode : *native*

Depuis le noeud

- Compiler avec l'option -mmic
- Utiliser l'outil micnativeloadex pour lancer le programme
- ...ou se connecter en ssh sur la carte

```
$ icc -ope
$ micnativ
```

```
 ...
Hello World from thread = 20
Hello World from thread = 10
Hello World from thread = 21
Hello World from thread = 10
Hello World from thread = 9
Hello World from thread = 12
Hello World from thread = 11
Hello World from thread = 39
...
```

Exemple Hello OpenMP avec offload

```
1  #include <omp.h>
2
3  #include <stdio.h>
4  #include <omp.h>
5
6  main ()  {
7
8  int tid;
9
10 #pragma offload target (mic:0)
11 #pragma omp parallel private(tid)
12  {
13
14    tid = omp_get_thread_num();
15    printf("Hello World from thread = %d\n", tid);
16
17 }
```

hello-offload.c

Exemple Hello OpenMP avec offload

```
1  #include <omp.h>
2
3  #include <stdio.h>
4  #include <omp.h>
5
6  main ()  {
7
8  int tid;
9
10 #pragma offload target (mic:0)
11 #pragma omp parallel private(tid)
12  {
13
14    tid = omp_get_thread_num();
15    printf("Hello World from thread = %d\n", tid);
16
17 }
```

Déporter le bloc suivant sur le mic0

hello-offload.c

Le mode : *offload - LEO*

Depuis le noeud

- Insérer des *pragma offload* dans le code
- Compiler uniquement avec l'option -openmp
- Lancer le programme sur le CPU

```
$ icc -openmp -mmic hello-offload.c -o hello-offload
$ export MIC_ENV_PREFIX=MIC
$ MIC_OMP_NUM_THREADS=240 ./hello-offload
$ ./hello-offload
```

Le mode : *offload - LEO*

Depuis le noeud

- Insérer des *pragma offload* dans le code
- Compiler uniquement avec l'option -openmp
- Lancer le programme sur le CPU

```
$ icc -ope                                    o-offload
$ export M
$ MIC_OMP_
$ ./hello-
```

```
 ...
Hello World from thread = 14
Hello World from thread = 46
Hello World from thread = 12
Hello World from thread = 104
Hello World from thread = 90
Hello World from thread = 23
Hello World from thread = 120
Hello World from thread = 201
...
```

Le mode : *offload - automatique*

- Concerne uniquement certaines routines MKL
- Aucun changement dans le code !
- Une variable d'environnement à exporter :
  export MKL_MIC_ENABLE=1

Les routines MKL

- Level-3 BLAS functions
  - `?GEMM` (for m,n $>$ 2048)
  - `?TRSM` (for M,N $>$ 3072)
  - `?TRMM` (for M,N $>$ 3072)
  - `?SYMM` (for M,N $>$ 2048)
- LAPACK functions
  - `LU` (M,N $>$ 8192)
  - `QR`
  - `Cholesky`

Le mode : *symmetric*

MPI sur CPU et Xeon Phi

Depuis le noeud

- Uniquement Intel MPI
- Compiler deux versions : une version CPU, une version MIC
- Export une variable d'environnement export I_MPI_MIC=1
- Lancer l'exécution avec mpiexec.hydra

```
$ mpiicc -mmic hello-mpi.c -o hello-mpi.mic
$ mpiicc hello-mpi.c -o hello-mpi.exe
$ export I_MPI_MIC=1
$ mpiexec.hydra -host node1-51 -n 16 ./hello-mpi.exe
:  -n 244 -host mic0 /Work/kmazouzi/hello-mpi.mic
```

Le mode : *symmetric*

MPI sur CPU et Xeon Phi

Depuis le noeud

- Uniquement Intel MPI
- Compiler deux versions : une version CPU, une version MIC
- Export une variable d'environnement export I_MPI_MIC=1
- Lancer l'exécution avec mpiexec.hydra

```
$ mpiicc -
$ mpiicc h
$ export I
$ mpiexec.
:  -n 244
```

```
 ...
Hello World from process = 16
Hello World from process = 133
Hello World from process = 140
Hello World from process = 234
Hello World from process = 1
Hello World from process = 0
Hello World from process = 33
Hello World from process = 62
...
```

```
o-mpi.exe
pi.mic
```

1 Introduction

2 Architecture

3 Modèles de programmation

4 Performances

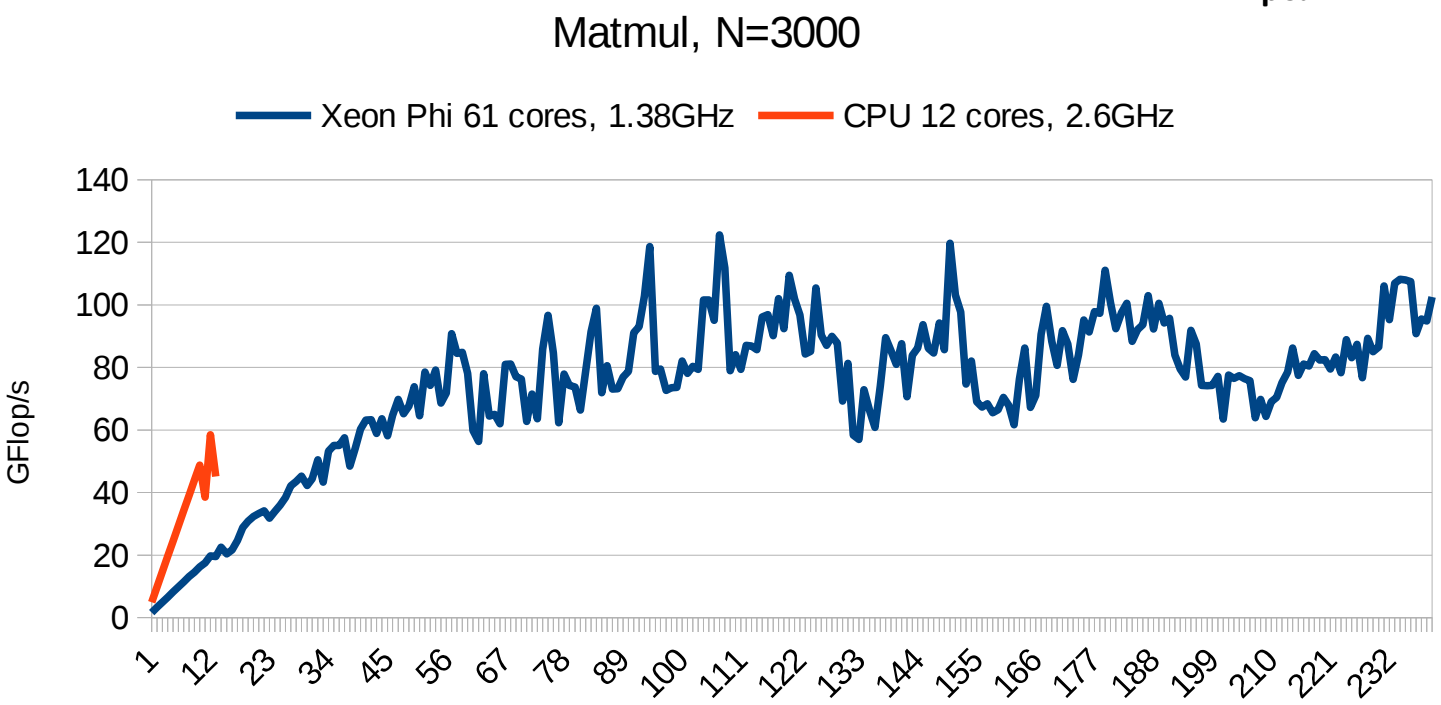
Bonnes pratiques

- Utiliser $N - 1$ cœurs, laisser 1 cœur pour le système
- Utiliser plusieurs Threads par cœurs (≤ 4)
- Placer les Threads sur les cœurs (*CPU Affinity*) avec la variable *KMP_AFFINITY*
  - valeurs possibles : *compact, balanced, scatter*
- Activer l'auto-vectorisation, l'option *-vec-report3* pour le diagnostique
- Aligner les données

Les outils d'analyse de performances sont limités à *Intel Vtune*

Matmul OpenMP sur Xeon Phi

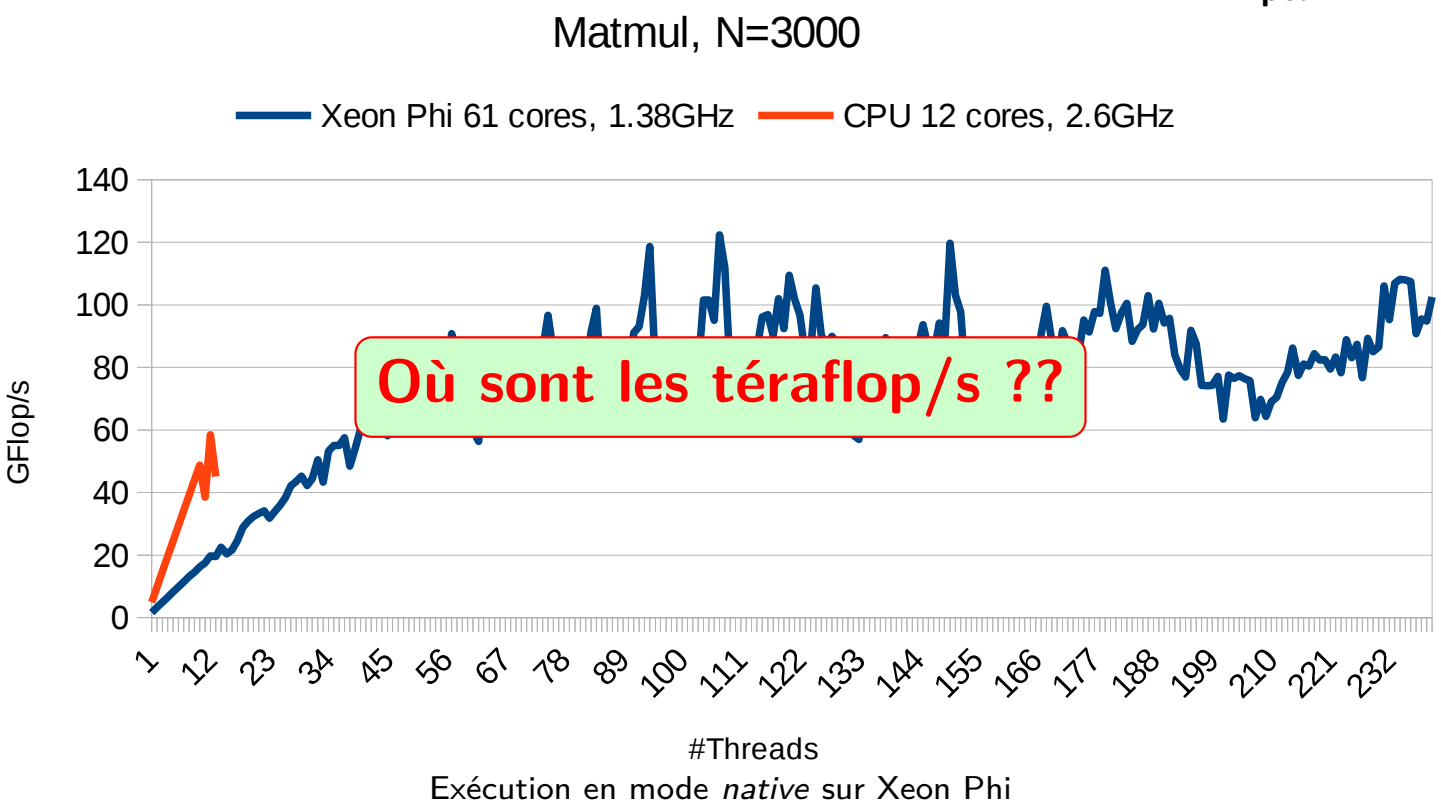
Rpeak = 1.2 TFlop/s



Exécution en mode *native* sur Xeon Phi

Matmul OpenMP sur Xeon Phi

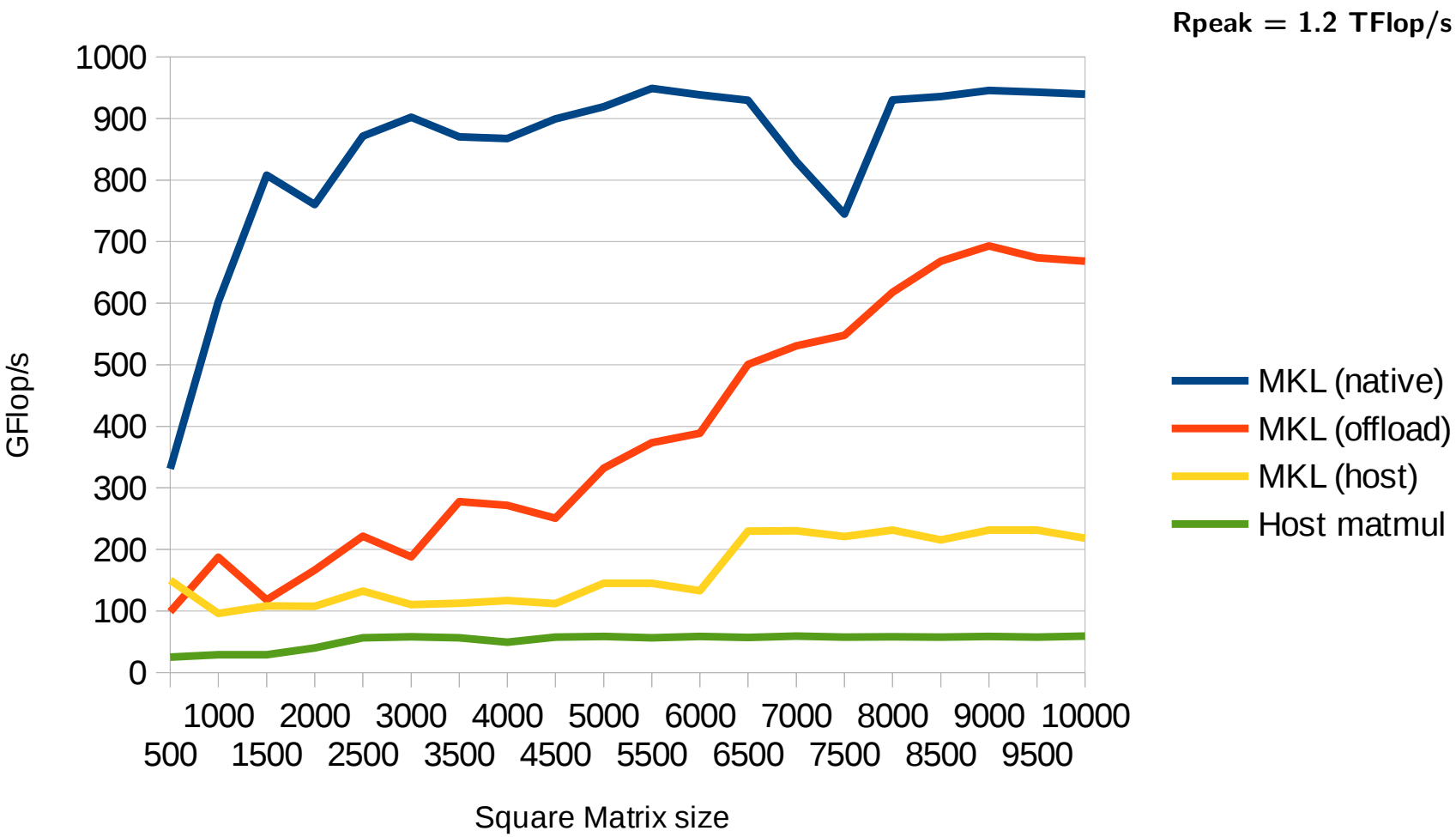
Rpeak = 1.2 TFlop/s



Exécution en mode *native* sur Xeon Phi

DGEMM sur Xeon Phi

- DGEMM : version optimisée, MKL, de Matmul
- 240 threads sur Xeon Phi, 12 threads sur CPU



Will My Code Run on Xeon Phi?

Yes

... but that's the wrong question

– Will your code run best on Phi?, or

– Will you get great Phi performance without additional work?

(The answer is most likely **NO**)

Source : Intel FAQ

Xeon Phi au mésocentre

File d'attente SGE xphi.q (cluster lumière)

Ressources

- Intel(r) Xeon(r) CPU e5-2630 v2 @ 2.60GHz (Ivybridge)
- 12 coeurs, 128G de mémoire
- 3 cartes 5110P, 60 x 1.053GHz, 8Go, 1 TFlop/s (DP)
- 1 cartes 7120P, 61 x 1.38GHZ, 16Go, 1.2 TFlop/s (DP)

Questions ?